Attorney Docket No. 081862.P149
Express Mail No.: EL236788713US

UNITED STATES PATENT APPLICATION

FOR

SCHEDULING NETWORK TRAFFIC USING MULTIPLE LOGICAL SCHEDULE TABLES

INVENTORS:

Radesh Manian
Shirish K. Sathe

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

SCHEDULING NETWORK TRAFFIC USING MULTIPLE LOGICAL SCHEDULE TABLES

Field of the Invention

This invention relates to computer networks. In particular, the invention relates to traffic scheduling.

The Background of the Invention

Technologies for computer networks have advanced at a fast rate to accommodate the needs for efficient and reliable communication. Designs for computer networks are now becoming complex both in hardware and software. To reduce complexity, most computer networks are organized as a series of layers or protocols, each one built upon the one below it. The function of each layer is to provide certain services to the higher layers, shielding those layers from the specific and detailed implementation of these services.

A network architecture typically follows some reference model to maintain universality and standardization. Examples of important reference models include the broadband Integrated Services Digital Network (B-ISDN) Asynchronous Transfer Mode (ATM), the Open System Interconnection (OSI), and the Transmission Control Protocol/Internet Protocol (TCP/IP) reference models. In general, a network architecture has the following layers: application, transport, network, data link, and physical. In these layers, the actual data transmission takes place in the physical layer.

An ATM wide area network (WAN) typically consists of multiple ATM interfaces. In the outbound direction, the network traffic shaping is performed on a per virtual connection basis. Each ATM interface supports multiple classes of service (COSes), each having its own unique shaping requirements. A typical network system supports multiple ATM interfaces with speeds ranging from 1.5 Magabits per second (Mbps) (e.g., T1) to 155 Mbps (e.g., OC-3).

To support a single ATM interface, many ATM segmentation and reassembly (SAR) processors are commercially available. These SAR processors typically have a fixed number of hardware schedule tables (HSTs) that are used for traffic shaping according to some scheduling algorithm. These HSTs can support a number of COSes.

When the system configuration has service requirements that exceed the number of COSes supported by the HSTs, a single SAR processor cannot be used. Multiple SAR processors can be used to satisfy the service requirements. However, this increases hardware complexity and creates a burden to the system management.

SUMMARY OF THE INVENTION

A method and apparatus are described for scheduling traffic in a network.

The method comprises (a) dividing a hardware schedule table into N logical schedule tables which are separated by table delimiters; and (b) assigning an identifier corresponding to a connection in the network in a scheduling table which is one of the N logical schedule tables.

Other features and advantages of the invention will be apparent from the detailed description and drawings provided herein.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings in which like references indicated similar elements which:

Figure 1 shows a system in which one embodiment of the invention can be practiced.

Figure 2 shows a conversion of a hardware schedule table into a set of logical schedule tables.

Figure 3A shows an example of an assignment of a connection identifier in an entry of a schedule table.

Figure 3B shows an example of an assignment of a connection identifier in an entry of a schedule table by wrapping around the schedule table.

Figure 4 shows an example of an assignment of a connection identifier in an entry of a logical schedule table set.

Figure 5A shows an example of an assignment of schedule tables to a system configuration.

Figure 5B shows an example of using logical schedule tables for the hardware schedule tables.

Figure 6 shows a flowchart for a process to allocate schedule tables to a system configuration.

Figure 7 shows a flowchart for a process to schedule traffic using logical schedule tables.

DETAILED DESCRIPTION

A method and apparatus are described for scheduling traffic in a network. A hardware schedule table is divided into N logical schedule tables separated by table delimiters. An identifier corresponding to a connection request is assigned to one of the N logical schedule tables.

In the following description, the description refers to the ATM model and the network segmentation and reassembly processor is merely an interface example. It is contemplated that the technique is applicable to other models, buses, or network architectures with similar characteristics.

ATM technology provides a high level of services to data communication. The basic idea of ATM is to transmit information in small, fixed-size packets called cells. The cells are 53 bytes long, of which 5 bytes are header and 48 bytes are payload. The advantages of using cell-switching technology in ATM includes flexibility in accommodating both constant and variable rate traffic, high speeds of data transfers, and broadcasting ability.

An ATM network architecture includes a physical layer, an ATM layer, and an ATM adaptation layer. The physical layer deals with the physical medium. The ATM layer deals with cells and cell transport, including congestion control. The ATM adaptation layer provides segmentation and re-assembly (SAR) of packets of cells. The UTOPIA is an interface between an upper ATM layer module and a physical layer module. To manage the network traffic, a SAR processor is used to perform segmentation and reassembly of packets of cells. In addition, many SAR processors also have schedules to schedule connections at specified time slots for traffic shaping.

Figure 1 is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 include N processors 105_1 to 105_N, a host bus 110, a host bridge chipset 120, a system

memory 130, a peripheral bus 140, a mass storage device 150, a network segmentation and reassembly (SAR) processor 155, a network interface bus 160, and K network interface processors 180_1 to 180_K.

Each of the N processors 105_1 to 105_N represents a central processing unit of any type of architecture, such as complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), explicitly parallel instruction set computing (EPIC), or hybrid architecture. Various embodiments of the invention could be implemented on a multi-processor or single processor computer system.

The host bridge chipset 120 includes a number of interface circuits to allow each of the N processors 105_1 to 105_N access to the system memory 130 and the peripheral bus 140. The system memory 130 represents one or more mechanisms for storing information. For example, the system memory 130 may include non-volatile or volatile memories. Examples of these memories include flash memory, read only memory (ROM), or random access memory (RAM). The system memory 130 includes program 132, data 134, and a logical scheduler 136. Of course, the system memory 130 preferably contains additional software (not shown), which is not necessary to understanding the invention. The logical scheduler 136 includes program, code, and data to perform the scheduling to shape the traffic for the network 190. The logical scheduler 136 mainly controls the network SAR processor 155. The logical scheduler 136 may also reside in a local memory accessible to the network SAR processor 155. The co-processor on the network SAR processor 155 may also execute the code corresponding to the logical schedule 136.

The peripheral bus 140 is a bus that allows each of the N processors 105_1 to 105_N communicate with the peripheral devices coupled to the peripheral bus 140. Examples of the peripheral bus include the peripheral components

interconnect (PCI). The mass storage device 150 includes CD ROM, floppy diskettes, and hard drives. The SAR processor 155 receives and transmits messages to the network 190. The SAR processor 155 includes a scheduler 158. The scheduler 158 performs scheduling to shape traffic on the network and includes a number of hardware schedule tables (HSTs). The network SAR processor 155 interfaces to the network interface processors 180_1 to 180_K via a network interface bus 160 such as the UTOPIA.

The scheduler 158 is typically part of a traffic management unit in the SAR processor 155. The traffic management unit provides traffic shaping for several service categories. Examples of these service categories include constant bit rate (CBR), variable bit rate (VBR), unspecified bit rate (UBR), and available bit rate (ABR). The categories may also include real-time (RT) or non real-time (NRT) support. The scheduler 158 schedules traffic by assigning connection identifiers from virtual circuit channels (VCCs) in HSTs, or scoreboards. The scheduler 158 sequences through the HSTs in a circular fashion to schedule the specified (e.g., CBR, VBR, ABR) traffic. When a connection identifier is assigned to an occupied entry in a HST, the scheduler 158 finds the next available entry in the HST, including wrapping around to the beginning of the HST, and assigns the connection identifier in the next available entry.

The logical scheduler 136 controls the scheduler 158 by creating a set of logical schedule tables (LSTs) from the HSTs to increase the support scope of the scheduler 158 when necessary such as when the number of service requests exceeds the available HSTs.

Each of the network interface processors 180_1 to 180_K is a device to provides interface to a network 190. The network 190 is connected to several communication ports to support various communication protocols and speeds such as OC3, DS3, and multiple T1/E1 ports.

When implemented in software, the elements an embodiment of the present invention are essentially the code segments to perform the necessary tasks. The program or code segments can be stored in a processor readable medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable ROM (EROM), a floppy diskette, a compact disk CD-ROM, an optical disk, a hard disk, a fiber optic medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, Intranet, etc.

Figure 2 shows a conversion 200 of a hardware schedule table into a set of logical schedule tables. The conversion 200 converts a hardware schedule table (HST) 210 into a logical schedule table (LST) set 220.

The HST 210 typically a part of the scheduler 158 in Figure 1. The HST 210 includes N entries for the assigned connection identifiers. The N entries are numbered 0, 1, . . . N-2, N-1.

The LST set 220 includes three LSTs: LST0 212, LST1 214, and LST2 216. As is known by one skilled in the art, the number of LSTs can be any number as required by the system configuration. The three LSTs 212, 214, and 216 are separated by three table delimiters 213, 215, and 217. A table delimiter is used to separate two adjacent LSTs and is typically an unused entry. Each table delimiter corresponds to a LST. For example, the delimiters 213, 215, and 217 correspond to the LST0 212, LST1 214, and LST2 216, respectively. The LST0 212

has P entries numbered from 0 to P-1. The LST1 214 has Q entries numbered from 0 to Q-1. The LST2 has R entries numbered from 0 to R-1. The LSTs in the LST set 220 operate independently of one another.

Let S be the total number of LSTs in the LST set 220. Let P_i be the size of the LST_i in the LST set 220 where i = 0, . . ., S-1. Let N be the size of the HST 210. Each LST has a table delimiter. Typically, the table delimiter corresponds to an unused entry. The size of the table delimiter may be more than one entry if necessary. When the size of the table delimiter is one entry, the following expression holds:

$$N = S + \sum P_i \qquad i = 0, \ldots, S-1 \qquad (1)$$

In one embodiment, when N and S are powers of 2, the P_i's are the same and are determined as:

$$P_i = (N/S) - 1 \qquad (2)$$

For example, when N = 2048 and S = 4, then each LST has P_i = 255 entries. The use of power-of-2 sizes facilitates the assignment of the connection identifiers in the schedule table.

Figure 3A shows an example of assignment 300A of a connection identifier in an entry of a schedule table. The assignment 300A shows a schedule table 310 before and after the assignment.

The schedule table can be either HST or LST. In this example, the schedule table 310 has N entries. Entry numbers 2, 3, 4, 5, 6, 8 are occupied and have been assigned the connection identifiers 100, 150, 75, 82, 23, and 10, respectively. Suppose connection identifier 50 is scheduled at entry or slot number 4 according to some traffic scheduling algorithm. However, entry number 4 has been occupied by the connection identifier 75. Therefore, the connection identifier 50 has to be assigned to another entry.

Since scheduling is time sensitive, it is desirable to assign the requested connection as close as possible to the originally assigned entry. The simplest method is to assign the connection identifier in the next available entry starting from the originally assigned entry going toward the end of the schedule table. In the example shown in Figure 3A, the next available entry is at entry number 7. Therefore, the connection identifier 50 is assigned to entry number 7.

Figure 3B shows an example of an assignment 300B of a connection identifier in an entry of a schedule table by wrapping around the schedule table. The assignment 300B shows a schedule table 320 before and after the assignment.

The schedule table can be either HST or LST. In this example, the schedule table 320 has N entries. Entry numbers 0, 1, 2, 4, N-7, N-6, N-4, N-3, N-2, and N-1 are occupied and have been assigned the connection identifiers 125, 171, 110, 82, 14, 13, 56, 92, 46, and 118, respectively. Suppose connection identifier 50 is scheduled at entry or slot number N-3 according to some traffic scheduling algorithm. However, entry number N-3 has been occupied by the connection identifier 92. Therefore, the connection identifier 50 has to be assigned to another entry.

The next available entry is found by going from the originally assigned entry number to the end of the schedule table and wrapping around to the beginning of the table and continuing to the rest of the table. In this example, the next available entry is entry number 3. Therefore the connection identifier 50 is assigned to entry number 3.

Figure 4 shows an example of an assignment 400 of a connection identifier in an entry of a logical schedule table set. The assignment 400 shows a LST set 410 before and after the assignment.

The LST set 410 includes three LSTs: LST0 412, LST1 414, and LST2 416 having P, Q, and R entries, respectively. The LST0 412, LST1 414, and LST2 416

have table delimiters 422, 424, and 426 at entry numbers P, Q, and R, respectively. The LSTs 412, 414, and 416 operate independently of one another. The assignment of a connection identifier in one LST is independently of that in another LST. In each LST, the assignment follows the same procedure as illustrated in Figures 3A and 3B.

In the example shown in Figure 4, the connection identifier 50 is originally assigned to the entry number P-3 of the LST0 412 according to some traffic scheduling algorithm. The entry P-3 of the LST0 412, however, has been occupied by the connection identifier 14. Therefore the connection identifier 50 has to be assigned to another entry in the LST0 412.

The scheduler 158 (shown in Figure 1) finds the next available entry in the HST containing the three LSTs 412, 414, and 416 and assigns the connection identifier in the found available entry. Then, the scheduler 158 returns the entry number of this available entry. The logical scheduler monitors this returned entry number to determine if this available entry is within the corresponding LST. If this available entry is within the corresponding LST, then the scheduling is complete. If this available entry is outside the corresponding LST, the logical scheduler 136 (shown in Figure 1) re-assigns the connection identifier to an entry within the corresponding LST. Since each LST has a delimiter corresponding to an unused or available entry, there is always an available entry at the end of each LST. The scheduler 158, therefore, can never assign the connection identifier past the LST boundary into another LST territory. Knowing the locations of the table delimiters, the logical scheduler 136 can determine if the connection identifier needs to be re-assigned. If the connection identifier is assigned in the table delimiter of the current LST by the scheduler 158, the logical scheduler 136 re-assigns the connection identifier by requesting the scheduler 158 to schedule the connection at the beginning of the LST.

In the example shown in Figure 4, the next available entry after the entry number P-3 is the delimiter 422 at entry number P. The scheduler 158 assigns the connection identifier 50 in entry number P and returns the entry number. The logical scheduler 136 monitors the returned entry number and recognizes that this entry number corresponds to the location of the delimiter 422 of the LST0 412. The logical scheduler 136 then removes the connection identifier 50 from the entry number P and finds another available entry in the LST0 412 starting from the beginning of the LST0 412 with the help of the hardware scheduler. In the example shown in Figure 4, the entry number 1 is unoccupied and therefore the connection identifier is assigned in the entry number 1.

An advantage of this scheme is that the scheduling is performed by the scheduler 158 and is assisted by the logical scheduler 136 only when the assignment goes beyond the corresponding LST. In most cases, the scheduler 158 can find an available entry within the current LST. Since the scheduler 158 is typically a hardware co-processor within the SAR processor 155 (shown in Figure 1), the processing speed of the scheduling is fast. The logical scheduler 136 only occasionally interferes with the scheduling when the next available entry coincides with the table delimiter of the current LST. The advantage of broadening the support scope of the scheduler by the LSTs far outweighs the minor performance slow processing time by the logical scheduler 136.

Figure 5A shows an example of an assignment 500 of schedule tables to a system configuration. The assignment 500 assigns the HSTs in the scheduler 550 to a system configuration 510 according to the support requirement 530.

The system configuration 510 includes an OC-3 port 512, a backplane 514, four T1/E1 ports: port 0 516, port 1 518, port 2 522, and port 3 524. The support requirement 530 includes four classes of service (COSes) for the OC-3 port 512, two COSes for the backplane 514, and four COSes for each of the four T1/E1

ports 516, 518, 522, and 524. The scheduler 550 has eight HSTs: HST0 552, HST1 554, HST2 556, HST3 558, HST4 562, HST5 564, HST6 566 and HST7 568. Each of the HSTs has 2048 entries.

The total number of schedule tables needed for this system is 22 (4 for the OC-3 port 512, 2 for the backplane 514, and 16 for the four T1/E1 ports 516, 518, 522, and 524). Since there are only 8 HSTs in the scheduler 550, there is not enough HSTs to support the system configuration. Therefore, logical schedule tables are to be created.

In this example, the OC-3 port 512 and the backplane have high priority and it is desirable to fully support the services. Therefore, 6 HSTs in the scheduler 550 are used to support the OC-3 port 512 and the backplane. The remaining 2 HSTs are used to support the four T1/E1 ports 516, 518, 522, and 524 using logical schedule tables.

For the OC-3 port 512, the HST0 552, HST1 554, HST2 556, and HST3 558 support the COS0 (e.g., CBR and rt VBR traffic), COS1 (e.g., VBR traffic), COS2 (e.g., ABR traffic), and COS3 (e.g., UBR traffic), respectively. For the backplane, the HST4 and HST5 support the COS0 (e.g., CBR and rt VBR traffic) and COS1 (e.g., rest of traffic), respectively. The HST6 566 supports the T1/E1 port 0 516 and T1/E1 port 1 518. The HST7 568 supports the T1/E1 port 2 522 and T1/E1 port 3 524. To provide complete support for all four COSes, each of the HST6 566 and HST7 568 is divided into eight LSTs.

Figure 5B shows an example of using logical schedule tables for the hardware schedule tables shown in Figure 5A.

The HST6 566 is divided into eight LSTs having the same size: LST0 570, LST1 571, LST2 572, LST3 573, LST4 574, LST5 575, LST6 576, and LST7 577. The LST0 570, LST1 571, LST2 572, and LST3 573 support the T1/E1 port 0 516 for the COS0, COS1, COS2, and COS3, respectively. The LST4 574, LST5 575,

LST6 576, and LST7 577 support the T1/E1 port 1 518 for the COS0, COS1, COS2, and COS3, respectively.

The HST6 568 is divided into eight LSTs having the same size: LST0 580, LST1 581, LST2 582, LST3 583, LST4 584, LST5 585, LST6 586, and LST7 587. The LST0 580, LST1 581, LST2 582, and LST3 583 support the T1/E1 port 2 522 for the COS0, COS1, COS2, and COS3, respectively. The LST4 584, LST5 585, LST6 586, and LST7 587 support the T1/E1 port 3 524 for the COS0, COS1, COS2, and COS3, respectively.

Figure 6 shows a flowchart for a process 600 to allocate schedule tables to a system configuration.

Upon START, the process 600 determines if the hardware schedule tables in the scheduler can support the system configuration sufficiently (Block 610). This determination can be based on a number of system criteria, such as priorities and classes of services. If yes, the process 600 is terminated because no logical schedule tables are necessary. Otherwise, the process 600 allocates some of the HSTs for a portion of the network services (Block 620). This allocation is based on priorities in the system configuration.

Then the process 600 divides the remaining HSTs into logical schedule tables (LSTs) such that the total number of LSTs sufficiently support the remaining system configuration (Block 630). The LSTs are separated by table delimiters. Each of the table delimiters corresponds to at least one unused entry in the LSTs. Then the process 600 is terminated.

Figure 7 shows a flowchart for a process 700 to schedule traffic using logical schedule tables.

Upon START the process 700 selects a LST to respond to an identifier of a connection (Block 710). The identifier is initially assigned to an entry in the selected LST. Then the process 700 determines if the initial entry for the identifier

is occupied (Block 720). If not, the process 700 assigns the identifier to the initial entry (Block 730) and the process 700 proceeds to Block 765. If the initial entry is occupied, the process 700 locates the next available entry in the LST and assigns the identifier in this available entry (Block 740). Then the process 700 returns the entry number of this available entry (Block 745).

Next, the process 700 determines if this entry coincides with the delimiter of the current LST (Block 750). If not, the process 700 goes to Block 765. If the entry coincides with the delimiter of the current LST, the process 700 removes the identifier from the LST delimiter location (Block 755). The process 700 then assigns the identifier in the next available entry starting from the beginning of the current LST (Block 760).

Next, the process 700 determines if all the identifiers have been scheduled in the LSTs (Block 765). If no, the process 700 goes to the next identifier (Block 770) and goes back to Block 710. Otherwise, the process 700 is terminated.

A technique has been described to schedule traffic in a network using logical schedule tables. The technique divides a hardware schedule table into a number of logical schedule tables separated by table delimiters. An identifier corresponding to a connection request is assigned to an entry in a logical schedule table. The technique allows a hardware schedule table to support multiple physical interfaces.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.